

Bioinformatik

Alignmentvarianten

Ulf Leser

Wissensmanagement in der
Bioinformatik



Motivation

- Grundlegende Gesetzmäßigkeit der Bioinformatik

Hohe Sequenzähnlichkeit bei DNA, RNA
und Proteinen heißt in der Regel
ähnliche Funktion bzw. Struktur

Alignment

- Definition

- Ein *(globales) Alignment* zweier Strings A, B ist eine Untereinanderanordnung von A und B , jeweils mit beliebigen zusätzlichen Leerzeichen ($_$)
 - Achtung: Zeichen dürfen matchen oder nicht
- Der *Alignmentsscore* eines Alignment ist die Anzahl von Leerzeichen und Mismatches
- Der *Alignmentabstand* zweier Strings A, B ist der minimal mögliche Alignmentsscore aller Alignments der beiden Strings

- Beispiele

– A_TGT_A	A_T_GTA	_AGAGAG	AGAGAG_
AGTGTC_	_AGTGTC	GAGAGA_	_GAGAGA

Score: 3

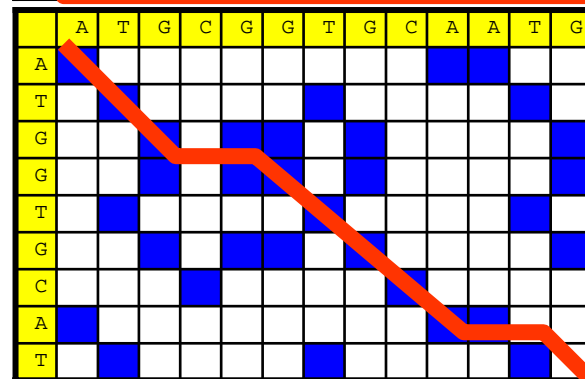
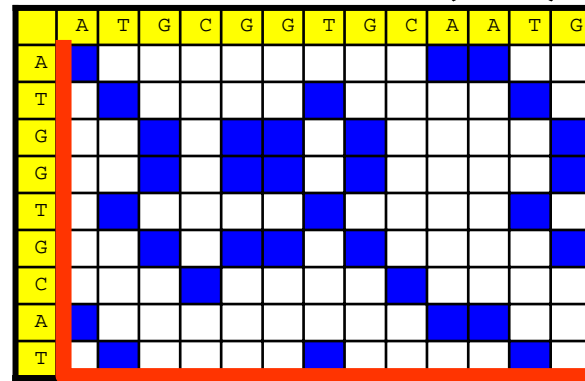
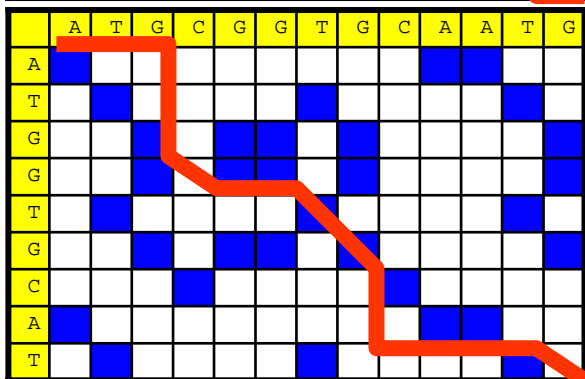
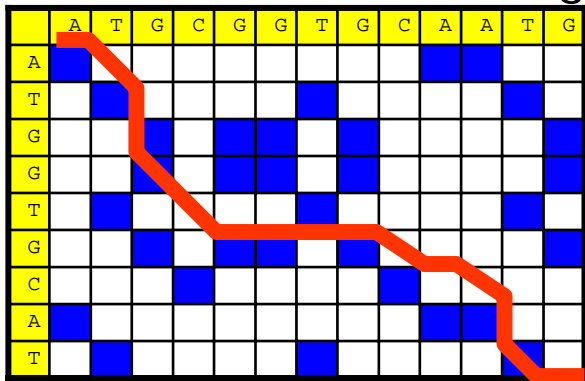
5

2

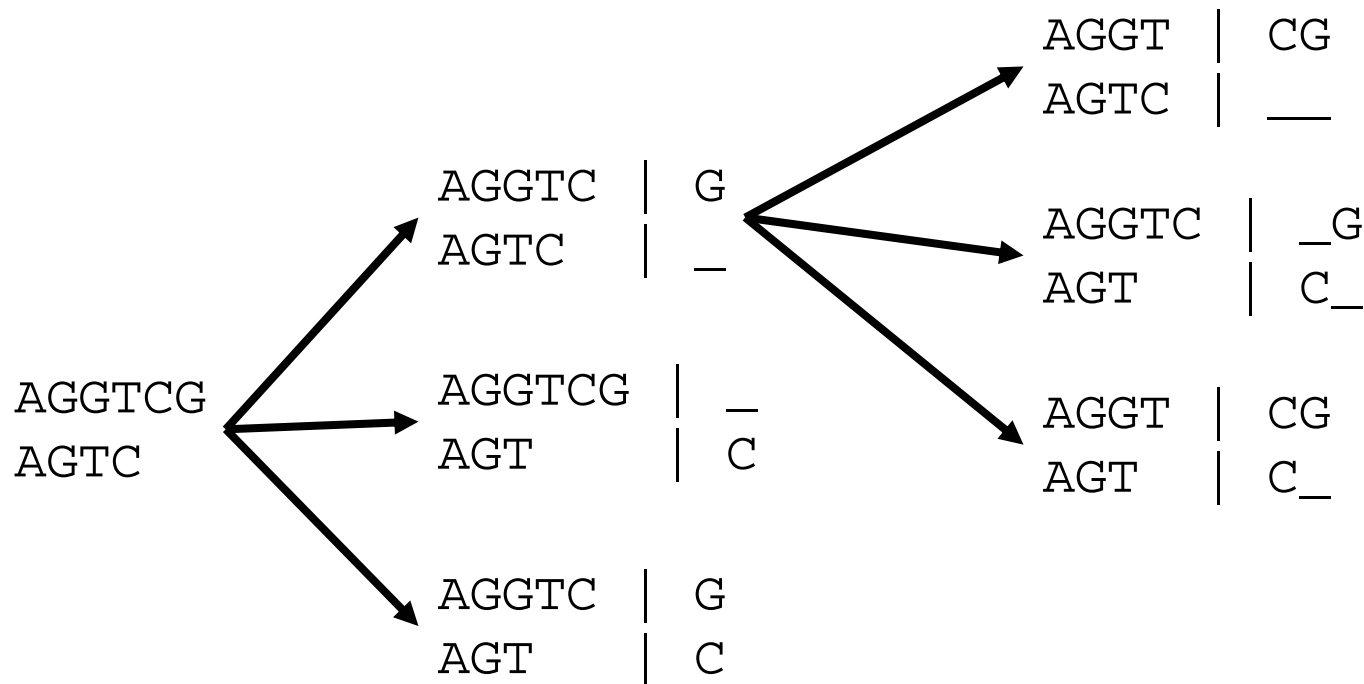
2

Alignments und Dotplots

- Sei $|A|=m$, $|B|=n$
- Betrachte **Pfade** im Dotplot von $(1,1)$ nach (m,n)
 - Pfad startet in linker oberer Ecke
 - Erlaubte Schritte: nach rechts, nach unten und nach rechts-unten (diagonal)
 - Pfad: Zusammenhänge Menge von Schritten bis (m,n)



Rekursive Betrachtung von Alignments



Editabstand

- Definition

Gegeben zwei Strings A , B mit $|A|=n$, $|B|=m$

- *Funktion $\text{dist}(A,B)$ berechne den Editabstand von A , B*
- *Funktion $d(i,j)$, $0 \leq i \leq n$ und $0 \leq j \leq m$, berechne den Editabstand zwischen $A[1..i]$ und $B[1..j]$*

- Bemerkungen

- Jedes R kann durch $\{I,D\}$ ersetzt werden; also werden R bevorzugt
- Offensichtlich gilt: $d(n,m) = \text{dist}(A,B)$
- $d(i,j)$ dient zur rekursiven Berechnung von $\text{dist}(A,B)$

Rekursive Berechnung 1

- Wir betrachten die Berechnung von $d(i,j)$ für A,B
 - Wir haben die optimalen Editskript für $A[1..i_0]$ mit $B[1..j_0]$, $i_0 \leq i \wedge j_0 \leq j \wedge \neg(i_0 = i \wedge j_0 = j)$, berechnet
 - Wie kann das Editskript weitergeführt werden?
- Fallunterscheidung
 - 1. Insertion in A (oder Deletion in B)
 - Situation:
 ...I
 XXX_
 XXXT
 - Also benutzen wir ein Zeichen mehr von B
 - $d(i,j-1)$ ist der Editabstand von $A[1..i]$ zu $B[1..j-1]$
 - Symbolisiert durch die XXX
 - Damit: $d(i,j) = d(i, j-1) + 1$

Rekursionsgleichung

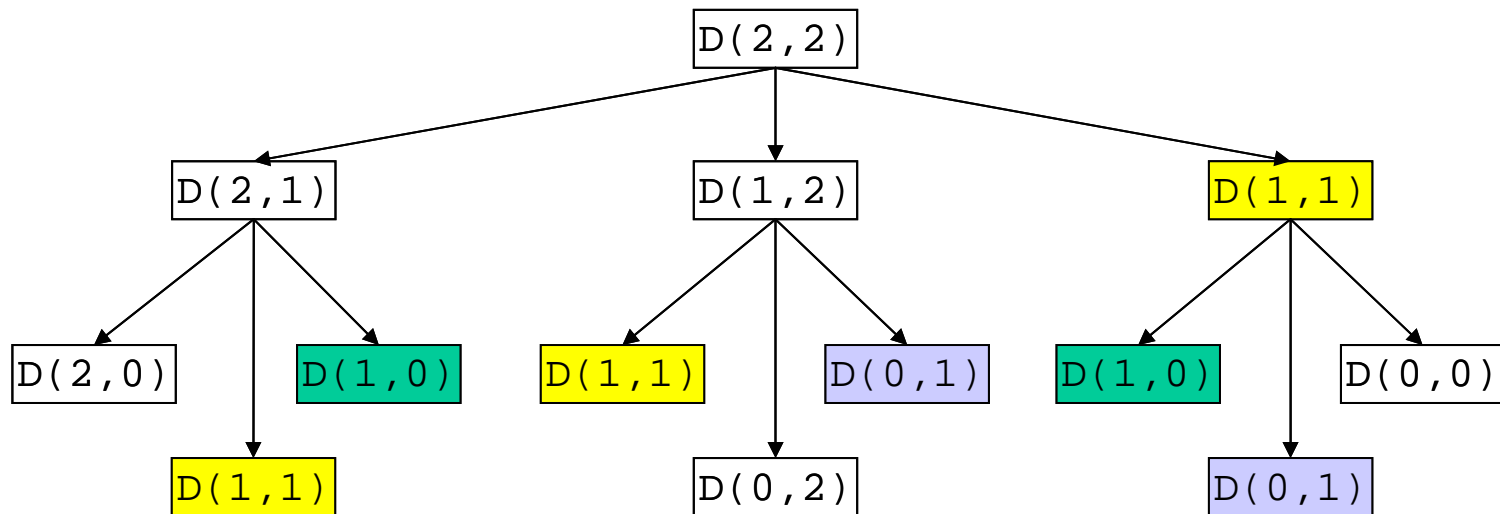
- Wir leiten das nächste Symbol im Editskript aus schon bekannten Editabständen ab
- Wir suchen das kürzeste Skript, also

$$d(i, j) = \min \left\{ \begin{array}{l} d(i, j-1) + 1 \\ d(i-1, j) + 1 \\ d(i-1, j-1) + t(i, j) \end{array} \right\}$$

$$t(i, j) = \begin{cases} 1: & \text{wenn } A[i] \neq B[j] \\ 0: & \text{sonst} \end{cases}$$

Sicher nicht optimal

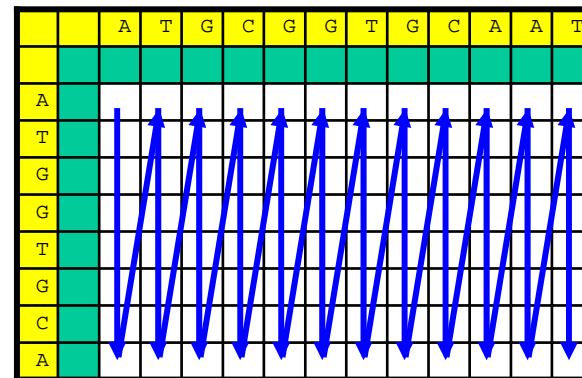
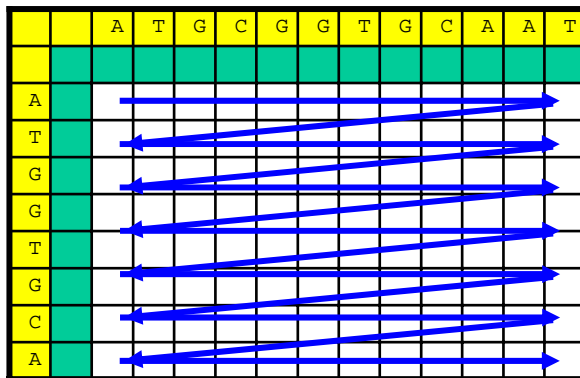
- Durch die Rekursionsgleichung werden viele Teillösungen mehrfach berechnet



- Es gibt nur $(n+1) * (m+1)$ verschiedene Aufrufe
- Wie kann man die redundanten Berechnungen sparen?

Tabellarische Berechnung

- Grundidee
 - Speichern der Teillösungen in Tabelle
 - Bei Berechnung Wiederverwendung wo immer möglich
- Aufbau der Tabelle: Bottom-Up (statt rekursiv Top-Down)
 - **Initialisierung** mit festen Werten $d(i,0)$ und $d(0,j)$
 - **Sukzessive Berechnung** von $d(i,j)$ mit steigendem i,j
 - Für $d(i,j)$ brauchen wir $d(i,j-1)$, $d(i-1,j)$ und $d(i-1,j-1)$
 - Verschiedene Reihenfolgen möglich



Vom Pfad zum Alignment

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

- Jeder Pfad von (n,m) nach $(1,1)$ ist ein optimales Alignment
 - Starte von $(1,1)$
 - Nach rechts: Deletion in A
 - Nach unten: Insertion in A
 - Diagonal: Match/Replace

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

ATGCGGT
ATG_G__

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

ATGCGGT
AT__GG_

Kürzeste Wege in Editgraphen

- Definition

Ein *Editgraph* für A, B mit $|A|=n$, $|B|=m$ ist

- Ein Graph mit $m \cdot n$ Knoten, beschriftet mit (i, j) für $0 \leq i \leq n$, $0 \leq j \leq m$
- Kanten und Gewichte
 - $(i, j-1) \rightarrow (i, j)$ mit Gewicht 1
 - $(i-1, j) \rightarrow (i, j)$ mit Gewicht 1
 - $(i-1, j-1) \rightarrow (i, j)$ mit Gewicht 0 gdw. $A[i]=B[j]$; sonst 1

- Es gilt

- Alle „leichtesten“ Wege sind optimale Alignments
- Umkehrung gilt auch
- Probleme sind aufeinander reduzierbar

Ähnlichkeit

- Definition

Gegeben Alphabet $\Sigma' = \Sigma \cup _$, Strings A, B über Σ' mit $|A| = |B| = n$

- Eine *Scoringfunktion* ist eine Funktion $s: \Sigma' \times \Sigma' \rightarrow \text{Integer}$
- Die *Ähnlichkeit* von A, B bzgl. der Scoringfunktion s ist

$$\text{sim}(A, B) = \sum_{i=1}^n s(A[i], B[i])$$

- Bemerkung

- Wir suchen natürlich wieder das Alignment, für das die Ähnlichkeit der beiden erweiterten Strings am größten ist

$$d(i, j) = \max \left\{ \begin{array}{l} d(i, j-1) + s(_, B[i]) \\ d(i-1, j) + s(A[i], _) \\ d(i-1, j-1) + s(A[i], B[j]) \end{array} \right\}$$

Beispiel

- AGGAG
- GAGA

Lösung

		a	g	g	a	g
	0	1	2	3	4	5
g	1	1	1	2	3	4
a	2	1	2	2	2	3
g	3	2	1	2	3	2
a	4	3	2	2	2	3

AGGAG_

__GAGA

AGGAG_

_G_AGA

AGGAG

GAGA_

GG_GAG

GAGA

...

Inhalt dieser Vorlesung

- End-Free Alignment
- Lokales Alignment
 - Smith-Waterman Algorithmus
- Alignment mit Gaps

Spezialfall Assembly

- Assembly beim Shotgun-Sequenzieren braucht keine Alignments der kompletten Strings



```
ACGTGATAGCTAGCTAG-----  
-----GATCGCTTGCAAGTATCTCTATAT
```

```
CAGGTTAGATGACCAGTAGCAGTGGCA  
-----GATTACCAGTAGGA-----
```

- Modellierung?

End-Free Ähnlichkeit

- Spaces am **Anfang und Ende sind umsonst**
 - Anfang: $d(i,0) = 0$, $d(0,j) = 0$
 - Wo ist der optimale „end-free“ Pfad?

	A	C	G	T	_
A	4	-2	-2	-2	-2
C		4	-2	-2	-2
G			4	-2	-2
T				4	-2
_					0

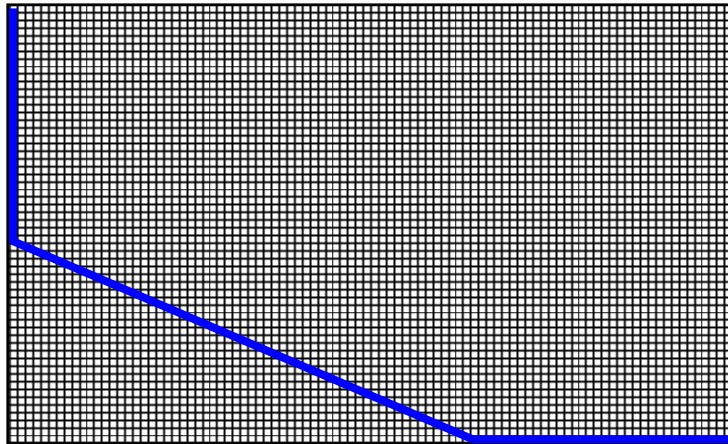
		T	T	G	G
	0	0	0	0	0
T	0	4	4	2	0
T	0	8	8	6	4
G	0	6	6	12	10

TTGG
TT_G

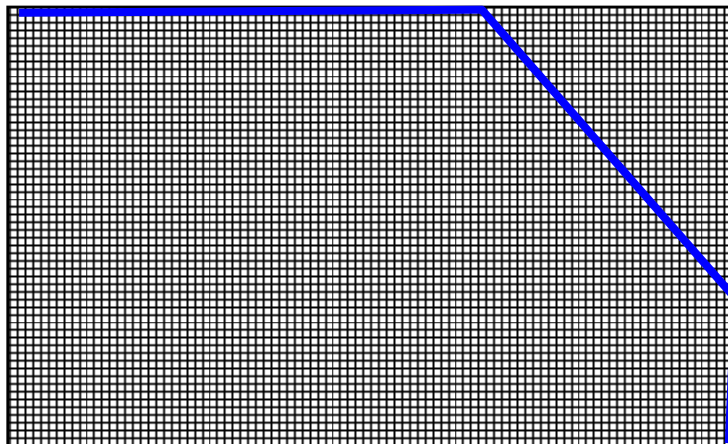
TTGG
TTG_

- Ähnlichkeit ist das **Maximum der Randzahlen**
 - Von dort bis (n,m) folgen nur Spaces – umsonst
 - Die (kostenlosen) Leerzeichen am oberen/linken Rand sind durch die geänderte Initialisierung berücksichtigt

End-Free Pfade



_____AAAA_AAAAAAAAAA_AAAAAAAAAA
 BBBBBBBBB__BBBB_BBBB_BBBBBBBB_____



AAAAAAAAAAAAAAAAAA_AA_AAAAAA_____
 _____BBBBBBBBB__BBBB_BBBB

? AAAAAAAAAAAAAAAAAA_AAAA_AAAAAAAAAA ?
 _____BBBBBBBBB__BBBB_____

Lokales und globales Alignment

- **Globales Alignments** betrachtet beide Sequenzen **komplett**
 - Größe zusammenhängender Matches spielt keine Rolle
- Biologie funktioniert anders
 - Evolution verschiebt Blöcke von Teilsequenzen
 - Blöcke bestimmen Funktion: Gene, Exons, Domänen, ...
- Also: Suche nach **gut passenden Teilsequenzen**

```
ACCCTATCGATAGCTAGGAAGCTCGATAATAACCGACCAGTAT
AGGAGTCGATAATAACATATAAGAGATAGAATATATTGATG
```

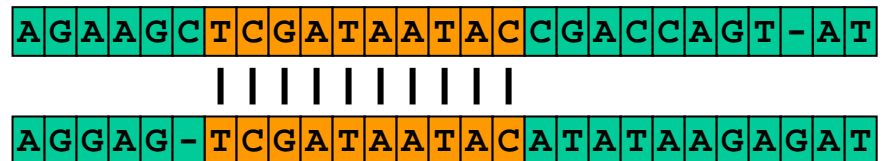
```
ACCCTATCGATA--GC-TAGGAAGCTCGATAATAACCGACCAGTAT-
|          | | | | |          | | | | |          | | | | |          | | |
A-GGAGTCGATAATAACATATAAG-A-GATAGAATATA-TTG-ATG
```

Lokales Alignment

- Definition. *Gegeben zwei Strings A, B.*
 - *Der lokale Ähnlichkeitsscore sim^* für A und B ist*

$$sim^*(A, B) = \max_{\forall a \in A, b \in B} (sim(a, b))$$

- *Das vom (globalen) Alignment von a und b induzierte Alignment von A und B heißt **lokales Alignment***
- Bemerkung
 - Unempfindlich gegen **unterschiedlich lange Sequenzen**
 - Wichtigkeit der „Blockung“ hängt von Scoringfunktion ab
- Beispiel
 - Lokales A. findet den identischen Substring
 - Das ist die **biologisch wichtige Information**



Berechnung lokaler Alignments

- Naiver Ansatz

- Aufzählen aller Substrings von A und in B
- Berechnung des optimalen Alignments aller Paare
- Auswahl des Paares mit der größten Ähnlichkeit

- Komplexität?

- Sei $|A|=|B|=n$
- Also $O(n^2)$ Substrings
- Damit $O(n^4)$ Paare
- Alignment ist $O(n^2)$
- Zusammen: $O(n^6)$

Länge des Substrings

n	n-1	n-2	...	1
1	2	3	...	n

Anzahl Substrings

- Das geht auch besser: $O(n \cdot m)$

Smith-Waterman Algorithmus

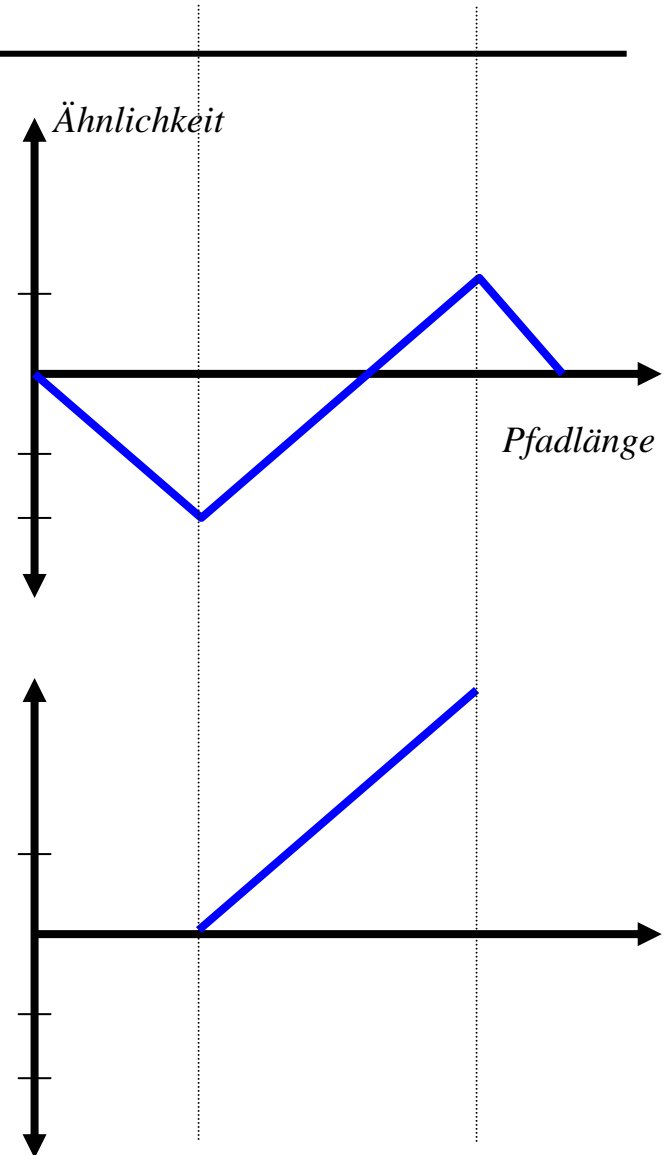
- Smith, Waterman: „Identification of common molecular subsequences“, J. Mol. Bio 147, 1981
- Annahme: Scoringfunktion mit positiven Werten für Matches und negativen Werten für alles andere
- Grundidee
 - Betrachten wir einen beliebigen Pfad
 - Eine Reihe von Matches erzeugt sukzessiv größere Scores
 - Bei Mismatch oder Insertion wird der Pfadscore wieder kleiner
 - So lange ein positiver Score übrig bleibt, kann noch was „Gutes“ entstehen
 - **Pfade mit negativem Score** kann man sofort vergessen

Match: +1
I/R/D: -1

Beispiel

		A	T	G	T	G	G
	0	-1	-2	-3	-4	-5	-6
G				-1			
T					0		
G						1	
A							0

		A	T	G	T	G	G
	0	-1	0	-3	-4	-5	-6
G				1			
T					2		
G						3	
A							0



Beispiel 2

Match: +1
I/R/D: -1

		A	T	G	T	C	G
	0	-1	-2	-3	-4	-5	-6
A	-1	1	0	-1	-2	-3	-4
T	-2	0	2	1	0	-1	-2
G	-3	-1	1	3	2	1	0

ATGTCG
ATG____
ATGTCG
AT____G
ATGTCG
A__T_G

➤ Drei Lösungen, alle mit gleicher Güte

		A	T	G	T	C	G
	0	0	0	0	0	0	0
A	0	1	0	0	0	0	0
T	0	0	2	1	1	0	0
G	0	0	1	3	2	1	0

ATGTCG
ATG____

➤ Eine Lösung – das lokale Alignment

Voraussetzung

- Definition

Gegeben Strings A , B und $i < n$, $j < m$.

Sei $a = A[1..i]$ und $b = B[1..j]$.

- *Das **lokale Suffixalignmentproblem** sucht die Suffixe a' von a und b' von b so, dass $\text{sim}(a', b')$ maximal ist*
- *Den maximalen Score bezeichnen wir als $v(i, j)$*

$$v(i, j) = \max_{\forall a' = A[x..i], b' = B[y..j]} (\text{sim}(a', b'))$$

- Bemerkung

- *Da $\text{sim}(\emptyset, \emptyset) = 0$, ist $v(i, j) \geq 0 \forall i, j$*

Folgerung

- Theorem

Gegeben Strings A, B

- *Der lokale Ähnlichkeitsscore sim^* für zwei Zeichenketten A, B ist*

$$sim^*(A, B) = \max_{i \leq n, j \leq m} (v(i, j))$$

- Beweis

- Offensichtlich

- Sprich: Wenn wir das lokale Suffixalignmentproblem lösen können, können wir auch das lokale Alignmentproblem lösen

Lösen des lokales Suffixalignmentproblems

- Theorem.
Gegeben Strings A,B. Dann gilt

$$v(i, j) = \max \left\{ \begin{array}{l} 0 \\ v(i, j-1) + s(_, B[i]) \\ v(i-1, j) + s(A[i], _) \\ v(i-1, j-1) + s(A[i], B[j]) \end{array} \right\}$$

- Beweisidee
 - Sehr ähnlich zum Beweis der ursprünglichen Rekursionsformel
 - Einzige Ausnahme ist die „0“ – der Reset
- Traceback
 - Starte beim **maximalen Wert in der Matrix**
 - **Nicht notwendigerweise am Rand**
 - Verfolge beliebigen Pfad bis zu einer **Zelle mit Wert 0**

Also?

- Lokales Alignment der Strings „ABCDE“ und „YXBZ“?

```
  _ABCDE_
YX_B_Z
```

- Lokales Alignment der Strings „ABCDE“ und „XYZ“
– Leeres Suffix

Wann lokales, wann globales Alignment

- Globales Alignment
 - Erfasst beide Sequenzen komplett
 - **Genauer Vergleich ähnlicher Sequenzen**
 - Charakterisierung von Protein-Familien
 - Bestimmung einer Consensus-Sequenz für MSA
- **Lokales Alignment**
 - Suche optimale Teilsequenzen
 - **Finden der interessanten Regionen in unbekanntem Sequenzen**
 - Z.B. unterschiedliche Spezies, unterschiedliche Gene, ...
 - Finden konservierter (=funktionaler) Subsequenzen
- End-Free Alignment?
 - Suche optimale Teilsequenzen so, dass **mindestens zwei Enden** von Sequenzen (egal von welchen) beteiligt sind

- Gapped Alignment

- Gap = Zusammenhänge Folge von Leerzeichen
- Bisher zählt jedes Leerzeichen einzeln
- Mehr Leerzeichen – linear mehr Kosten
- Ist das immer das richtige Modell?

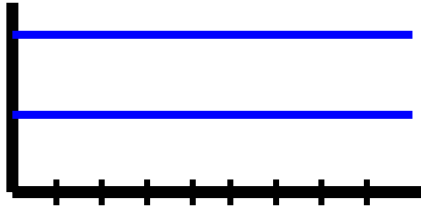
Gaps

- Evolution besteht nicht nur aus Punktmutationen oder einzelnen Baseninserts / -deletions
- Weitere Ereignisse: **Rearrangements**
 - Crossing-Over während Meiose (geschlechtliche Zellteilung)
 - „Versehentliche“ Duplikation von Sequenzen
 - Transposable Elements
 - Erscheint als Insert / Deletion eines Sequenzabschnitts
 - Ist aber nur ein evolutionäres Ereignis
- Sequenzblöcke sind wichtig
 - **Konservierte Regionen** in unterschiedlichen Genomen
 - Proteine setzen sich aus „**active sites**“ und variablen Zwischensequenzen zusammen

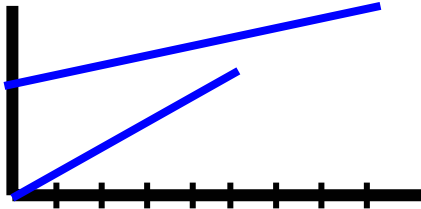
Bewertung von Gaps

- Sei $w(k)$ der Score eines Gaps der Länge k
- Funktionsklassen
 - **Konstanter** Gapscore c
 - $w(k) = c$
 - Score unabhängig von Gaplänge
 - **Linear** mit Kosten für Gapbeginn w_s und Gapfortsetzung w_f
 - $w(k) = w_s + w_f * k$
 - Bisheriges Modell nimmt $w_s=0$ und $w_f=1$ an
 - **Konvex** - Score wird nie kleiner, aber immer langsamer größer
 - $w(k) = f(k)$; mit $f'(k) > 0$ und $f''(k) \leq 0$
 - Beispiel: $w = \log(k)$
 - **Beliebiger** Gapscore - Score kann wachsen, fallen, oszillieren, ...
 - $w(k) = f(k)$; mit $f(k)$ beliebig

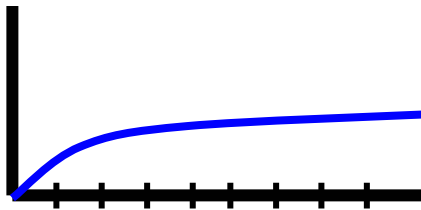
Klassen von Gapscorefunktionen



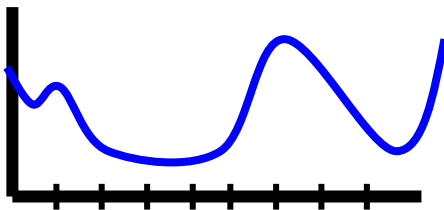
- Konstanter Gapscore



- Linearer Gapscore



- Konvexer Gapscore



- Beliebiger Gapscore

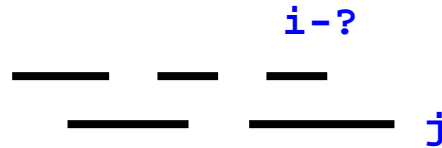
Auswirkungen auf Berechnung

- Je **flexibler die Gapscorefunktion**, desto komplexer die Berechnung
 - Konstant oder linear: $O(n * m)$
 - Konvex: $O(n * m * \log(m))$
 - Beliebig: $O(n * m^2 + n^2 * m)$
- Wir betrachten lineare und beliebige Gapscorefunktionen
 - Ähnlichkeit - Gapscores gehen also negativ ein
 - Notwendig: Erweiterung der Rekursionsgleichung
- Zunächst: **Beliebige Gapscores**

Erinnerung – Normalfall, Insertion in A

- Wir betrachten die Berechnung von $d(i,j)$ für A,B
- Fallunterscheidung
 - 1. Insertion in A (oder Deletion in B)
 - Situation:
 ...I
 XXX_
 XXXT
 - Also benutzen wir ein Zeichen mehr von B
 - Damit: $d(i,j) = d(i, j-1) - 1$
- **Voraussetzung** war, dass wir das optimale Alignment von $A[1..i]$ und $B[1..j-1]$ schon kennen
 - Durch das Einfügen eines „_“ wird der Score um 1 schlechter – egal, was das Alignment vor der aktuellen Position aussieht
 - Für beliebige Gapscores ist das „davor“ aber wichtig – Score ist unterschiedlich, je nachdem wie groß das Gap ist/wird

Beliebige Gapscores, Insertions in A



- Wir wollen immer noch das optimale Alignment von $A[1..i]$ und $B[1..j]$
- Und nehmen immer noch an, dass das letzte Zeichenpaar ein „_“ und $B[j]$ ist
 - Das letzte Zeichen von A, das mit einem Zeichen in B aligniert, liegt irgendwo links von $i-1$.
 - Danach kommen in A ein oder mehrere Leerzeichen – das Gap
 - Auswirkung der Länge des Gaps auf den Score ist „**unvorhersagbar**“
 - Alle möglichen Fälle betrachten – alle Gap-längen in A

Mögliche Alignments

$a_1 a_2 a_3 \dots a_{i-1}$		—
$b_1 b_2 b_3 \dots$		b_j

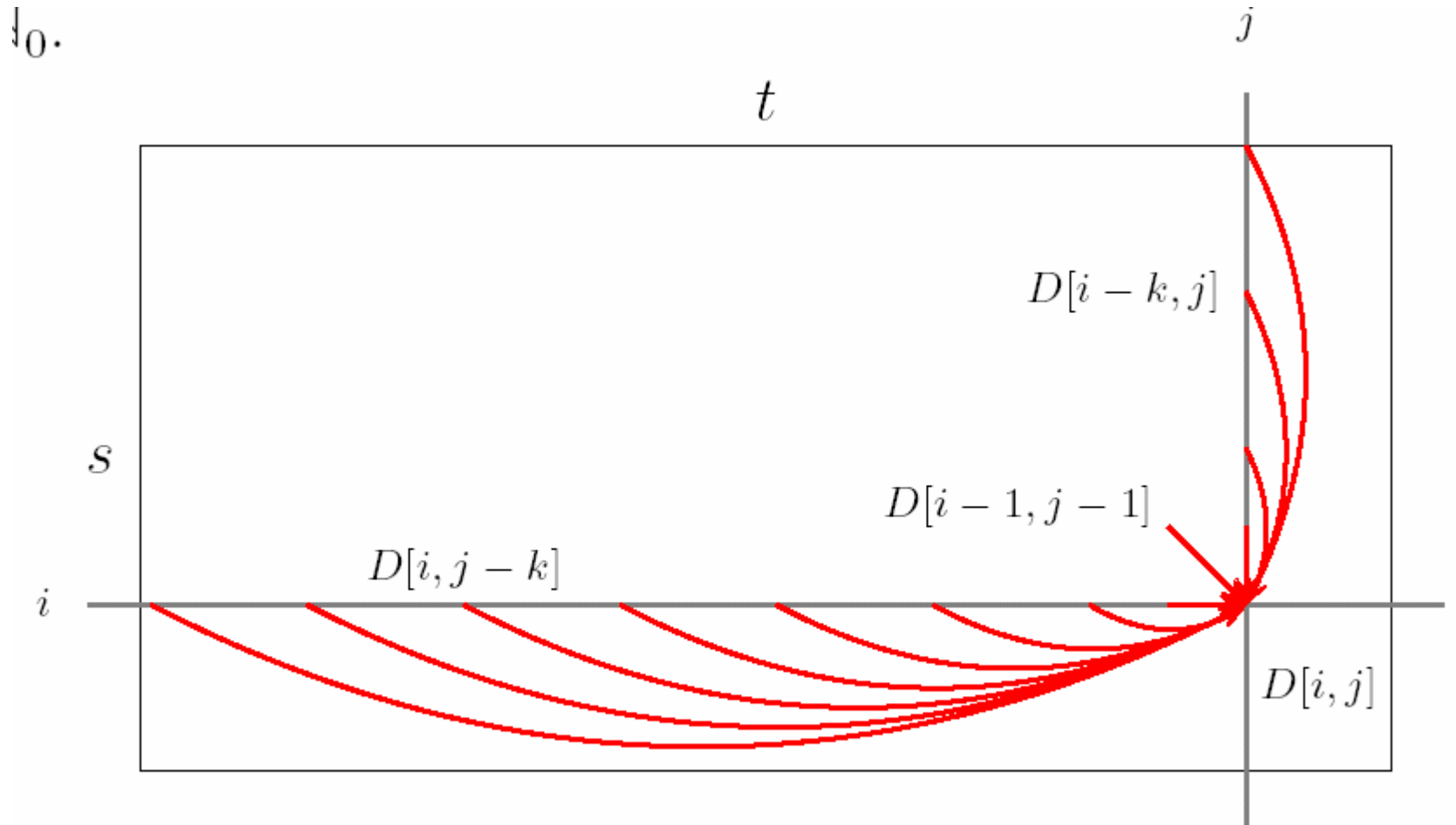
$a_1 a_2 a_3 \dots a_{i-2}$		—
$b_1 b_2 b_3 \dots$		b_j

Allgemeiner Fall

$a_1 a_2 a_3 \dots a_{i-k}$		—
$b_1 b_2 b_3 \dots$		$\dots b_j$

$$E(i, j) = \max_{1 \leq k \leq j-1} (V(i, k) - w(j - k))$$

Was müssen wir betrachten?



Drei Rekursionsgleichungen



$$E(i, j) = \max_{1 \leq k \leq j-1} (V(i, k) - w(j - k))$$



$$F(i, j) = \max_{1 \leq l \leq i-1} (V(l, j) - w(i - l))$$



$$G(i, j) = V(i - 1, j - 1) + s(A[i], B[j])$$

$$V(i, j) = \max(E(i, j), F(i, j), G(i, j))$$

Komplexität

- Theorem

Gegeben Strings A, B mit $|A|=n$, $|B|=m$. Für beliebige Gapscorefunktionen kann das optimale Alignment in $O(n^2m + nm^2)$ berechnet werden

- Beweis

- Wir berechnen Einträge einer Tabelle der Größe n, m
- In jeder Zelle berechnen wir 4 Werte: $E(i, j)$, $F(i, j)$, $G(i, j)$, $V(i, j)$
- G und V sind konstant
- Für E und F müssen die Zellen $(i-1, j-1)$, $(i, 1 \dots j-1)$ und $(1 \dots i-1, j)$ betrachtet werden
- Für fixes i (eine Spalte füllen) betrachten wir also \sum_j für $1 \leq j \leq n-1$ Zellen. Das ist $O(n^2)$
- Für fixe j (Zeile füllen) ist das $O(m^2)$
- Daraus folgt: $O(n \cdot m^2 + m \cdot n^2)$

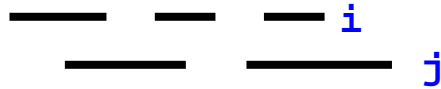
Lineare Gapscorefunktionen

- Beliebige Gapscorefunktionen
 - $w(k)$ kann mit steigendem k kleiner, größer oder unverändert werden/bleiben
 - Wir müssen deshalb alle möglichen Gap-längen an jeder Position ausprobieren
- Was muss man bei **linearen Funktionen** ausprobieren?
- Welche möglichen Fälle gibt es?

Lineare Gapscorefunktionen

- Der Score der Gaps wird mit wachsender Länge immer größer
 - Und zwar jedes Mal um einen konstanten Faktor w_f
- Eine Besonderheit sind aber die **Gapanfänge**
 - Die haben zusätzliche Kosten w_s
- Damit sind für Gaps in A zwei Fälle zu betrachten
 - Neues Gap in A: kostet $w_s + w_f$
 - Gap in A wird fortgesetzt: kostet w_f

Fälle



- Zwei Fälle

- Gap beginnt in A an Position i

$$E(i, j) = V(i, j - 1) - w_f - w_s$$

- Gap in A wird fortgesetzt

$$E(i, j) = E(i, j - 1) - w_f$$

- Zusammen

$$E(i, j) = \max(V(i, j - 1) - w_f - w_s, E(i, j - 1) - w_f)$$

Unterschied

$$E(i, j) = \max(V(i, j-1) - w_f - w_s, E(i, j-1) - w_f)$$

- Vorsicht
 - E sind die Kosten bis zu einer Zelle, wenn als letztes eine Insertion ausgeführt wurde
 - Entsprechend: F sind die Kosten bis zu einer Zelle, wenn als letztes eine Deletion ausgeführt wurde
- Wir müssen uns **E, F, V in jeder Zelle merken**
 - Bei beliebigen Gapscores ist das nicht notwendig
 - Da muss man so oder so die ganze Reihe ablaufen
- Platzbedarf steigt also
 - Um konstanten Faktor

Rekursionsgleichungen



$$E(i, j) = \max(V(i, j-1) - w_f - w_s, E(i, j-1) - w_f)$$



$$F(i, j) = \max(V(i-1, j) - w_f - w_s, F(i-1, j) - w_f)$$



$$G(i, j) = V(i-1, j-1) + s(A[i], B[j])$$

$$V(i, j) = \max(E(i, j), F(i, j), G(i, j))$$

Komplexität

- Theorem

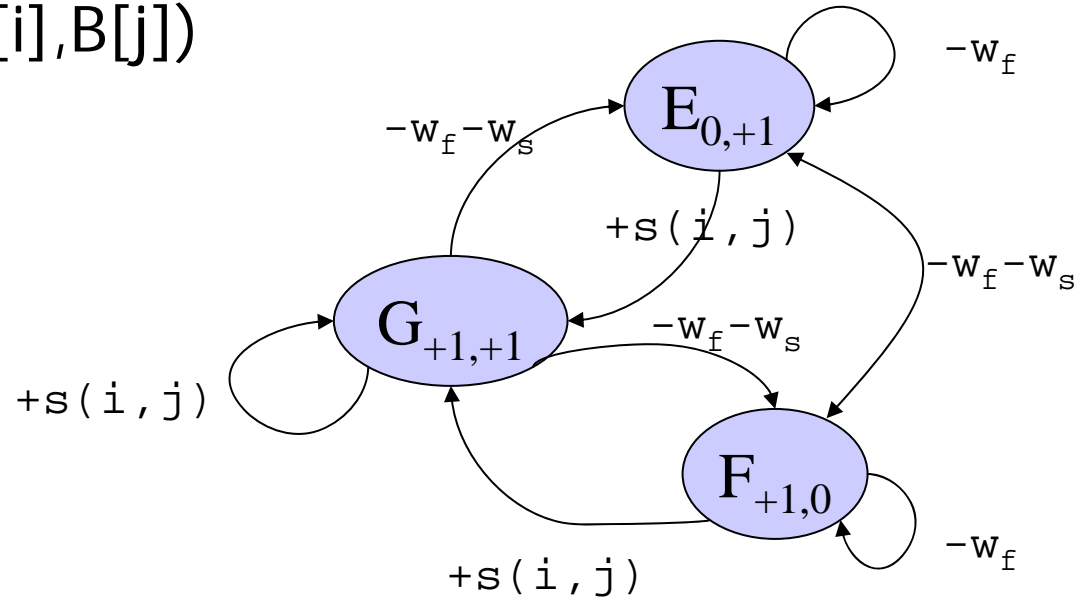
*Gegeben Strings A, B mit $|A|=n$, $|B|=m$. Für **lineare Gapscorefunktionen** kann das optimale Alignment in $O(n*m)$ berechnet werden*

- Beweis

- Wir berechnen in jeder Zelle die Werte $E(i,j)$, $F(i,j)$, $G(i,j)$
- Für die Berechnung jeder Zelle müssen nur konstant viele andere Zellen betrachtet werden
- Daraus folgt: $O(n*m)$
- qed.

Gap scores und endliche Automaten

- Sei $s(i,j) = s(A[i],B[j])$



- **Zustandsübergänge** sind Entscheidungen
 - Gap beginnen, Gap fortsetzen, Gap beenden, Matchen
- Optimale Alignments sind **Pfade mit maximaler Summe**
 - Beginnend bei $(0,0)$, endend bei (n,m)

Zusammenfassung

- Alignment ist vielfarbig
 - End-Free versus lokales versus globales Alignment
 - Gewichtete Editabstände (Match/Mismatch/Insert/Del)
 - Gap-Score Modelle (konstant/linear/konvex/beliebig)
 - Substitutionsmatrizen (später)
- Zusammenspiel ist kompliziert
 - Globales Alignment mit hohen Gapkosten erzeugt sehr kompakte Alignments (viele Mismatches)
 - Lokales Alignment mit niedrigen Gapkosten ähnelt globalem Alignment
- Konkrete Wahl eines Modells abhängig von der Aufgabe